高效能分散式深度學習系統之效能量測與分析技術

Performance Measurement and Analysis Techniques for High-Performance Distributed Deep Learning Systems

劉政岳、洪士灝
Cheng-Yueh Liu, Shih-Hao Hung

近年深度學習技術興起，加速人工智慧 (AI) 應用的發展。產學界爭相利用最新的高效能異質計算系統快速探索深度學習演算法的設計，並且解決重要的科學與工程的問題，乃至於衍生出許多商業和民生應用。因此，如何以大規模計算叢集，整合最新的硬體加速器，在最短時間內利用大數據訓練出精確的深度學習模型，即時部署到資料中心和物聯網，成為當前發展 AI 應用的過程中極為核心的系統基礎建設，其中包含許多複雜且具挑戰性的研究議題。本文介紹我們研究團隊所開發的效能測量和分析技術，對於建構與優化上述高效能分散式深度學習系統，提供重要的協助，可以讓系統開發人員分析各種類神經網路模型在普遍使用的深度學習框架進行散式訓練的效能，探討其中可能限制效能的因素，並且進一步對於系統架構設計、模型參數更新機制、編譯器選項等提供效能優化的建議。

Deep learning has been widely used to develop artificial intelligence applications recently. Academia and companies race to find new deep learning algorithms and solve important scientific/engineering problems, as well as develop applications for business and daily use with high-performance heterogeneous computing systems. Hence, the system infrastructure to accelerate the development and deployment of deep learning applications using a large-scale computing cluster with state-of-the-art accelerators is not only critical, but also contains many complexes, challenging research opportunities. In this article, we introduce our research work on performance measurement and analysis techniques, which provide essential information to help construct the aforementioned system infrastructure and enable the system designer to examine the factors that may affect the performance of various neural network models on mainstream deep learning frameworks. We show how our tools can be used to investigate on performance bottlenecks and optimize the system performance by adjusting the the system architecture, the parameter update mechanism, and the compiler options.

一、深度學習框架及大數據中介軟體

近年來深度學習 (deep learning, DL) 在人工智慧 (AI) 發展領域裡扮演舉足輕重的角色，許多公司與學術單位相繼推出開源的軟體框架 (software framework)，例如 Caffe，Torch，MXNet，TensorFlow 及 CNTK 等 DL 框架[1, 2]，便利使用者開發及佈署 AI 應用程式。許多深度學習應用必須仰賴大數據，因此上述的框架經常結合普遍用於處理大數據的 Hadoop 及 Spark 開源中介軟體，在計算叢集 (computing cluster) 上運作，以提高效能[3]。然而，由於資料遽增及應用領域多樣化，加上系統硬體針對大數據與深度學習不斷推陳出新，中介軟體為了更具通用性，其原始碼日趨龐大且複雜。若要在目前的深層軟體疊堆 (deep software stack) 上對系統效能進行優化，提升運算速度，工程人員必須借重效能分析工具，而且在使用此類工具時，不僅得留意觀察者效應 (observer effect)，更需要重視分析工具本身的時間耗損 (overhead)，方能有效分析大數據系統及中介軟體[4-7]。

二、高效能人工智慧計算系統

要加快利用深度學習處理大型數據集 (例如 ImageNet dataset)，或是將訓練完成的類神經網路模型用以提供高效能的即時智慧服務，往往必須仰賴平行處理或是專用加速器 (accelerator)。當前的高效能異質計算系統以高速網路 (例如頻寬可達 100 Gbps 的 Infiniband) 串聯多台電腦成為計算叢集，其中每台電腦可能配備數十核心的中央處理器 (CPU)、數千核心的繪圖處理器 (GPU)、可程式化的晶片 (FPGA)、以及固定功能的加速晶片 (ASIC) 來執行計算工作。由於採用數種處理器，因此稱為異質計算 (heterogeneous computing)，這些處理器的特性概述如下：

- CPU 適合執行作業系統，管理應用程式存放在記憶體與儲存系統的資料，以及與其他電腦透過網路通訊，因此通常做為存放深度學習神經網路的參數伺服器 (parameter server)。
- GPU 採用單指令流多數據流 (SIMD) 的模式，同時對數千筆資料以同一個指令進行同一類型的計算，由於 SIMD 模式相當搭配深度學習神經網路的計算模式，而且 GPU 的計算能量高出 CPU 數倍，因此通常將深度學習計算的工作送至 GPU 進行。
- FPGA 的可程式化特性能讓特定的函式 (function) 透過特殊的撰寫與編譯方式轉換成電路後執行，由於執行時不需要提取和解碼指令，因此效率高於 CPU 與 GPU，同時藉由流水線平行 (pipeline parallelism) 和資料平行 (data parallelism) 的架構設計，FPGA 有可能對深度學習推論或搜索排序等應用提供低延遲、高效率的解決方案。
- ASIC 則是進一步藉由更高度優化的硬體提升 FPGA 的低延遲及高效率的優勢，但是電路的功能固定，無法修改。目前最著名的深度學習 ASIC 是 Google 開發的 Tensor Processing Unit (TPU)[8]，已經被佈署在 Google 的資料中心用於加速深度學習應用，效率可達 CPU 或 GPU 的數十倍之多。

若要快速完成深度學習推論或搜索排序的工作，每一步的延遲必須盡可能降低[9]，但可能不利於處理器的效率。通常在使用 GPU 執行深度學習運算時，為了充分利用 GPU 的計算能力，會採取批次執行的方式，一次執行一批 (batch) 的工作。一批次的工作量 (batch size) 越高，處理器的效率越高，但單一工作的等待時間會增加，延遲可能高達毫秒量級，因此必須視應用的需求調整 batch size。

此外，高速計算叢集的各個電腦透過高速網卡交換資料時，執行通訊函式和協定的工作可能造成處理器的負擔，而網路傳輸的時間也會增加延遲，因此，如果系統支援高效率的資料傳輸模式，避開使用高負擔的 TCP/IP 通訊協定，減少資料在使用者空間 (user space) 與核心空間 (kernel space) 之間拷貝的次數，甚至利用網卡所提供 remote direct memory access (RDMA) 技術達成零資料拷貝 (zero data copy)，即有可能提昇傳輸效率及降低延遲。

總結上述，利用高效能計算系統加速深度學習應用時，必須有系統性地對應用進行效能分析，精準洞悉軟體與硬體之間的互動狀況，才能確實達成高效能的目標。

三、效能分析方法

面對深層軟體堆疊以及複雜的高效能異質計算架構，必須透過有系統的效能分析才能發現效能瓶頸和優化系統，因此效能分析可說是建構高效能系統或加速應用的「起手式」。透過效能剖析工具 (performance profiler)，不僅可以量測程式實際執行期間在特定系統結構上引發的硬體事件，也可以記錄程式碼本身在時域上的執行路徑 (execution code path)、熱點函式、記憶體使用效率等，進而了解程式特徵 (program characteristics) 以作出對應的優化策略。

效能分析工具之設計，通常著重在二方面：

(1) 效能資料蒐集 (trace collection)：效能分析工具蒐集資料的手段包含記錄硬體中斷及指令集代碼，啟動作業系統追蹤點 (tracepoint)，讀取 CPU 內建的效能計數暫存器 (performance counter)，執行期間插碼 (runtime binary instrumentation) 等。

(2) 效能資料表現 (representation)：在資料表現上，可分為「效能資訊統整 (performance profiling summary)」及「效能事件追蹤 (performance event tracing)」，前者是搜集完效能資料後，建構出函式與系統事件或硬體計數的組成分佈關係，例如程式的熱點分佈或 cache misses 在各個子函式之分佈；後者著重於呈現函式及硬體事件在時間軸上的相依性與因果性，藉由觀察這些變化而推導出可以優化的效能問題，例如每當某些函式被呼叫時皆會引發一定網路流量，而致使這些函式在時間軸上的足跡持續過久，則可透過檢查軟體是否採用異步傳輸或提高頻寬來縮短此整體時間。

四、常見的效能分析工具

現今以「效能資訊統整」見長的分析工具，有 perf，gprof，valgrind，Vtune，Vampir，Nsight 等軟體，然而這些工具在處理複雜的統計數據時，所提供的報告往往過於繁雜，經常無法直接提供分析人員所需要的數據。為此，Brendan Gregg 開發了專門把採樣到的堆疊軌跡 (stack trace) 轉化為直觀的視覺化表示，稱之為「火焰圖」(flame graph) [10]。火焰圖的 X 軸代表採樣總量，而 Y 軸代表棧深度。若在深層軟體堆疊上執行程式並進行熱點分

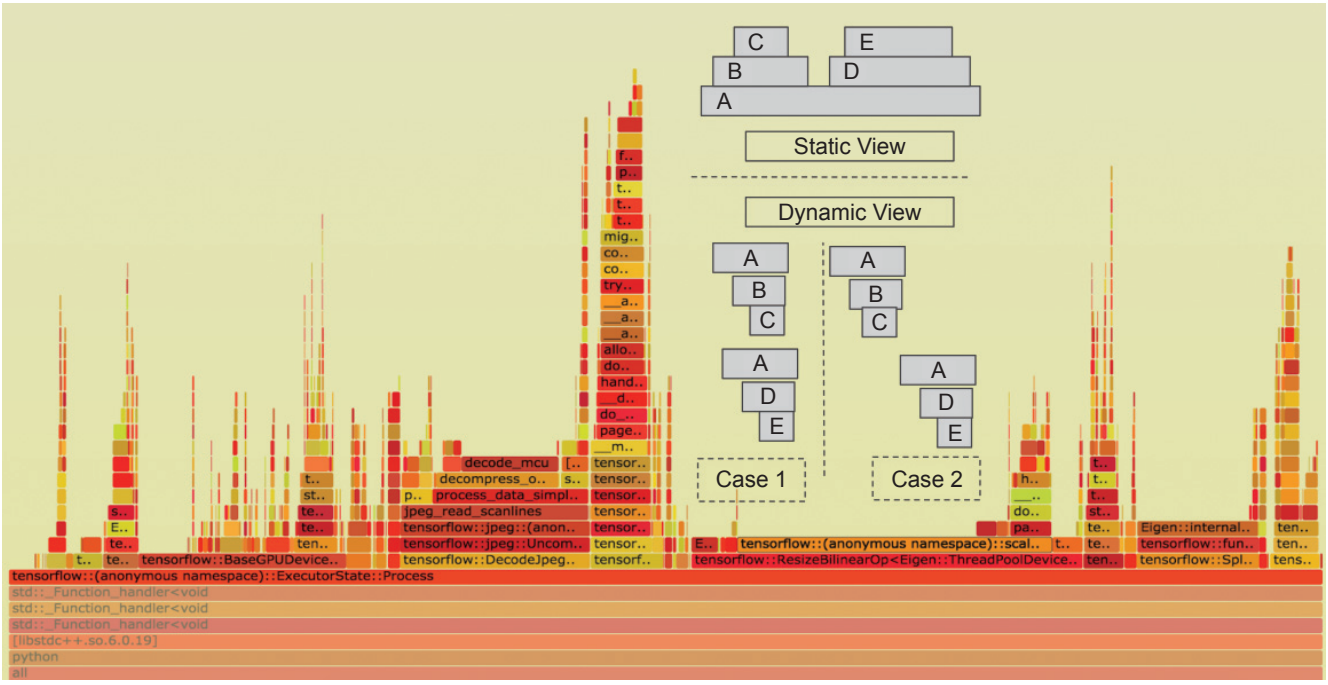


圖 1. TensorFlow 行 Resnet50 訓練之火焰圖。

析，探討某函式與另一函式，或是某函式與系統事件之因果關係時，火焰圖是一個非常有效率的工具。以圖 1 為例，它是以 TensorFlow 深度學習框架在單一節點多 GPU 卡之平台上執行 Resnet50 訓練的火焰圖。

每個框就代表了一個堆疊裡的函數，其寬度代表了所佔用 CPU 的總時間。通過火焰圖，分析人員可以輕鬆觀察到 TensorFlow 的函式呼叫結構及其佔用 CPU 的情況。然而，分散式或平行計算程式的效能問題如等待訊息或同步資料，皆和事件的時間順序有關，因此需要全部的事件蹤跡才能找到效能問題。再則，從上圖中可得知 A 函式包函了 B 及 D 兩大子函式，且 B 與 D 執行時間比例相等，但若就此斷然認為只要大幅地加速 B 函式即可讓 A 函式整體速度提昇 2 倍，便會犯了「只採用效能資訊統整而進行優化」的謬誤。

此時，以「效能事件追蹤」(performance-related event tracing) 見長的分析工具，便可以針對這種平行程式進行正確的效能剖析。在使用者空間裡，若採用基於執行期間插碼 (dynamic binary instrumentation) 的 VTune 及 Valgrind[11, 12]，可以深入分析使用者空間的程式碼及函數呼叫，但其觀察者效應可能造成量測誤差，而且無法分析與 GPU 等異質架構相關的效能事件。而涵蓋使用者空間及核心空間之經典全系統效能分析工具有 perf、lttng 及 vmpire[13-15]，皆是基於採樣方法，故有較小的觀察者效應。Linux 作業系統中提供的 ftrace 核心空間函數追蹤工具程式，能百分之分記錄下感興趣之核心程式碼軌跡，包含 context switch，wake-up/ready 等，只不過其觀察者效應因為不採用硬體指令而較大。

除了上述工具之外，我們可以透過在全系統模擬器 (full system emulator) 上插入虛擬效能計數器及追蹤工具而實現全系統事件追蹤，並且可透過校正模擬器之虛擬系統時間 (virtual timing device) 可彌補因插碼而造成的觀察者效應，例如 dist-gem5[16] 以及我們研究團隊所開發的 VPA[17]，然其缺點是分析時間成本巨大 (e.g. 僅數十 MIPS)，只適合分析鉅細靡宜且精簡的程式碼路徑，不適合進行大型分散式系統的研究[18]。

五、改進效能分析工具

然而上述的分析工具，且不論各有其優缺點，皆無法完整匯整所有異質系統之事件軌跡。例如 Linux perf 專司於 Linux 作業系統核心事件及 CPU 硬體效能計數暫存器採樣，ftrace 專司於 Linux 作業系統核心事件追蹤，Nvidia nvprof 專司於 GPU 事件及效能計數暫存器，tcpdump 專司於網路事件，Mallanox ibdump 專司 Mallanox InifiniBand 網路事件，vmstat/mpstat 專司於多核心系統上 process 與 virtual memory 之效能監測。若不整合多項工具，則難以萃取出全面資訊來分析深層軟體疊堆及異質計算系統之交互關係，而定位出潛藏的效能問題。

因此，我們研發 SOFA (Swarms of Functions Analysis) 這套基於「效能事件追蹤」的分析工具，整合了應用程式效能採樣、中介軟體及系統核心函式之符號追蹤記錄，系統活動監控記錄以及異質硬體效能資訊等，並以函式群 (swarm of functions) 之粒度進行效能分析與效能詮釋，進而提供效能優化建議，其軟體架構圖可見圖 2。

除此之外，SOFA 亦將上述效能資料融合為一張時間軸動態圖以進行視覺化分析，如圖 3 所示，其中 x 軸為絕對時間，單位為秒，y 軸為各類效能指標，其單位從 CPU/GPU 時間開銷、ibdump/tcpdump/vmstat 之系統事件計數到 vmstat/mpstat/nvidia-smi 系統資源使用率都有。為了在同一張圖裡採用近似尺度檢視，所有 y 值皆取其對數值 (log value) 後顯示在圖上。

透過關聯事件的時間標記匹配，我們將跨硬體時間標記系統作同步，進而讓各個函式及事件在時間軸上具有時域相關性。舉例而言，在 CPU 上記錄到的 nvvp 第一筆 cudaMemcpy 或 cudaMemset 函式呼叫，可作為絕對 GPU 時間基準 (absolute GPU time base)；接著以 nvprof --print-gpu-trace 所錄下的第一筆時間欄位作 GPU 時間相對平移 (relative GPU time offset)；有了基準及平移量，便可初步地同步 CPU 及 GPU 之時間標記系統。

此外，SOFA 提供兩大功能：(1) 以函式群分析為蹤跡作顯著化。(2) 異質計算事件關連。這兩

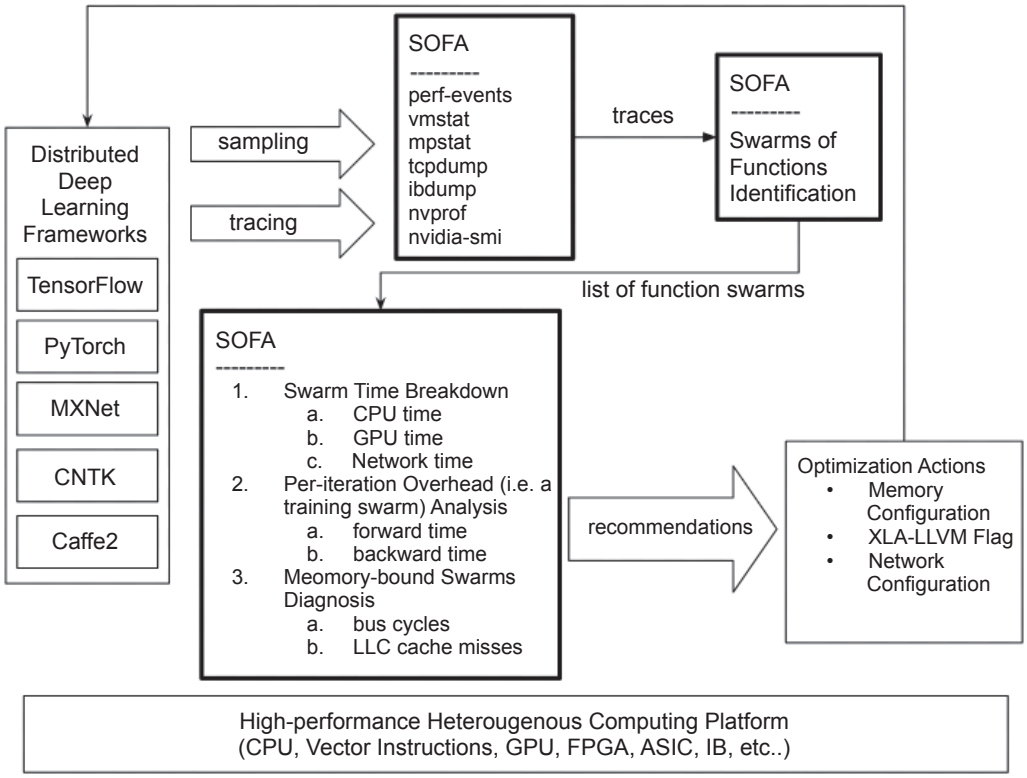


圖 2. SOFA 軟體架構圖。

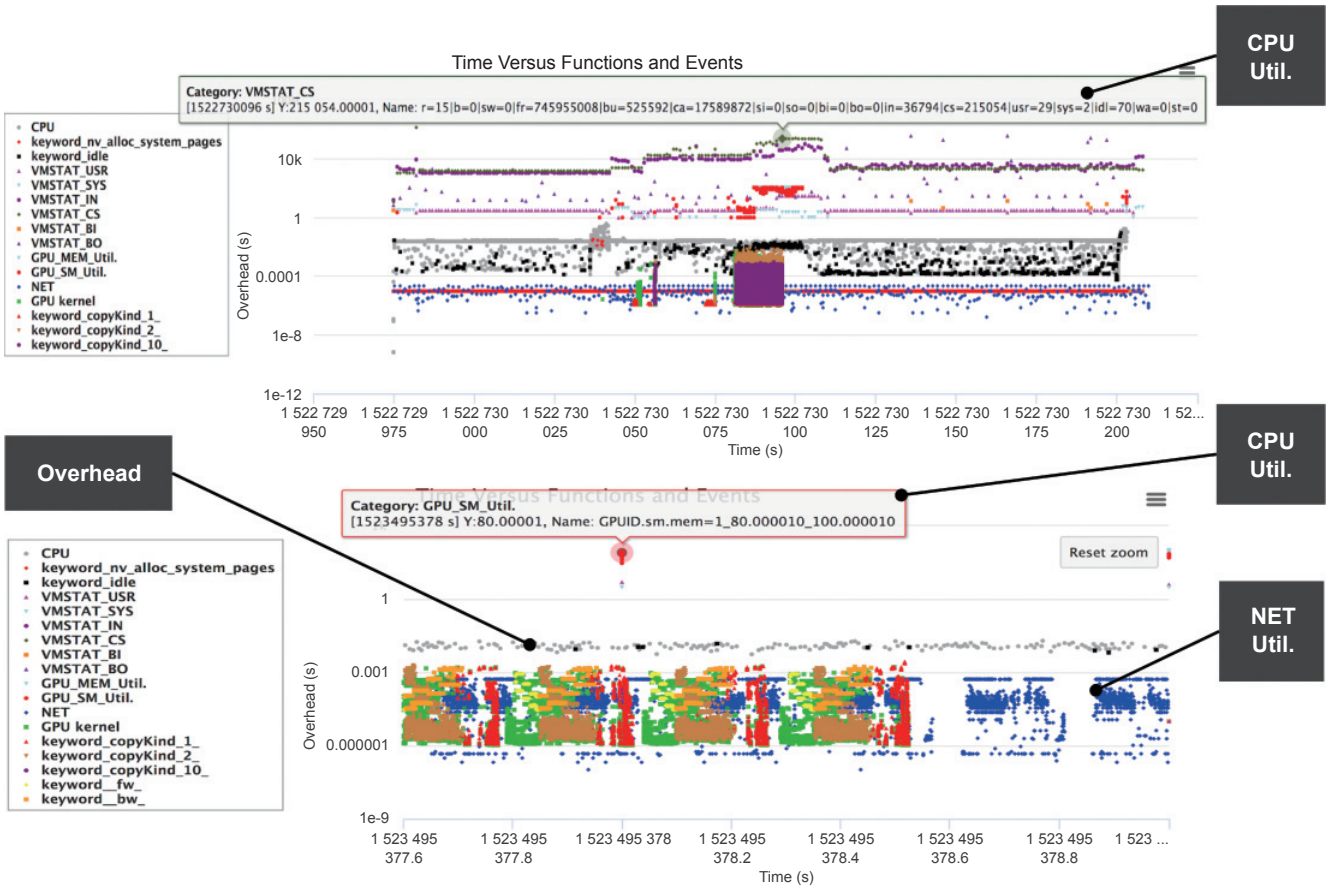


圖 3. SOFA 函式群及異質硬體效能分析。

大功能讓使用者能在日趨深層的軟體疊堆上解析效能。

・SOFA 特色 1：以函式群分析顯著化蹤跡

所有基於採樣的方析方法，都會有採樣率不足而造成漏失重要採樣點的隱憂，此時透過更大的觀察粒度 (granularity)，意即以「一整群函式」(swarm of functions) 的粒度去進行效能事件追蹤，往往可以看到更顯著的因果性 (causality) 及共時性 (concurrency) 。以函式群觀察出一個深度學習程式正在結束一個階段的參數同步為例，單看一個執行緒執行「鎖釋放 (lock release) 」函式，可能看不出什麼端倪，但看到一群執行緒作鎖釋放而產生大量鎖釋放的蹤跡，形成「鎖釋放群 (lock-release swarm)」，接著又發現鎖釋放群「周期性」地出現在一群 GPU 核心執行蹤跡 (GPU Kernel Traces) 及網路流量蹤跡 (network traces) 之後，便可推估這個鎖釋放群的出現時間長短可視為深度學習程式之參數同步的時間耗損，進而針對這期間出現的函式進行優化。

為辨識函式群的生成，我們透過蒐集中介軟體及系統核心函式的符號追蹤記錄，以關鍵字匹配的方式為各群函式在時間線圖 (timeline) 上染色，從而得到各種類型的函式群，例如 CPU 閒置 (keyword=indle) 群，Mellanox RDMA 通訊群 (keyword=mlx5)，圖像前處理群 (keyword=decode) 等等。越多同關語意的關鍵字被用以染色，所形成的群也會越巨大顯著。為了自動化生成更具有實際效能意義的群，我們未來打算利用機器學習的方法來達成，尤其是叢集算法 (clustering) 及遞歸神經網路 (recurrent neural networks, RNN) 等等有助於自動文字摘要 (auto text summerization) 的算法。

・SOFA 特色 2：異質計算事件關連

除了用 perf 分析程式及中介軟體之動態行為，由上層應用程式與中介軟體所引起的作業系統運行間效能計數 (runtime system counters) 變化，如網路傳輸流量 (network traffic)，進程切換 (context switch)，軟體中斷 (soft interrupt)，輸入／輸出等待 (I/O wait)，及記憶體置換次數 (swap counts) 等等，也是分析全系統效能時不可或缺的分析面向。因此，透過整合 vmstat，mpstat 及 tcpdump 等原生系統工具 (program trace)，我們便可支援上述全系統動態效能分析。

此外，異質架構系統上的各個子模組通常也會提供各自的效能記錄工具，例如 Nvidia GPU 的 nvprof 及 nvidia-smi，AMD GPU 的 coreXL，Mellanox IB 高速網卡的 ibdump。這些異質效能記錄軟體工具，讓我們可以存取異質架構效能計數器以便進行中介軟體與硬體效能之相關性分析。

・SOFA 的擴展性

SOFA 整合這些異質架構效能記錄工具，採模組化設計，未來若引入新式硬體，無論是 GPU、FPGA 或 ASIC，只要在 SOFA Record/Preprocess 等 python 腳本中加入各類硬體供應商所附之軟體工具 (utility) 之呼叫，即可將其效能資料融合至 SOFA 的時間軸動態圖中作分析。

特別一提的是，在接下來一系列的案例分析中，雖然僅涵蓋 GPU 平台，但工具的使用與分析方法對於其他的分散式異質計算平台仍具參考價值。目前多數異質架構採用 PCIe 介面連接加速器，而主機與加速器之間的資料通訊所造成的等待時間是主要效能瓶頸之一。因此若將個案中的 NVIDIA P100 GPU 換成 AMD 的 Vega Frontier GPU，或是比特大陸的 Sophon BM1680 SC1 ASIC，還是能夠依循著 SOFA 框架分別採取插件 CodeXL 及 BM Runtime 來進行分析。

・案例分析一：深度學習 HPC 伺服器之記憶體效能優化

深度學習高效能計算伺服器，常配備多 GPU 卡及高速內部網路 (interconnection)，例如 NVIDIA 推出之 DGX-1 及 HGX-1，兩者皆整合了 8 個 NVIDIA Tesla P100 GPU 與 NVLink/PCIe 混合式網狀架構互連，其架構可見圖 4。後者更進一步結合微軟 Project Olympus 開源設計以 NVIDIA NVLink™ 互聯技術與 PCIe 標準為基礎的轉換設計，實現各種 CPU 與 GPU 的機器配置，以利在雲端運算環境中大規模地且彈性化地佈署。

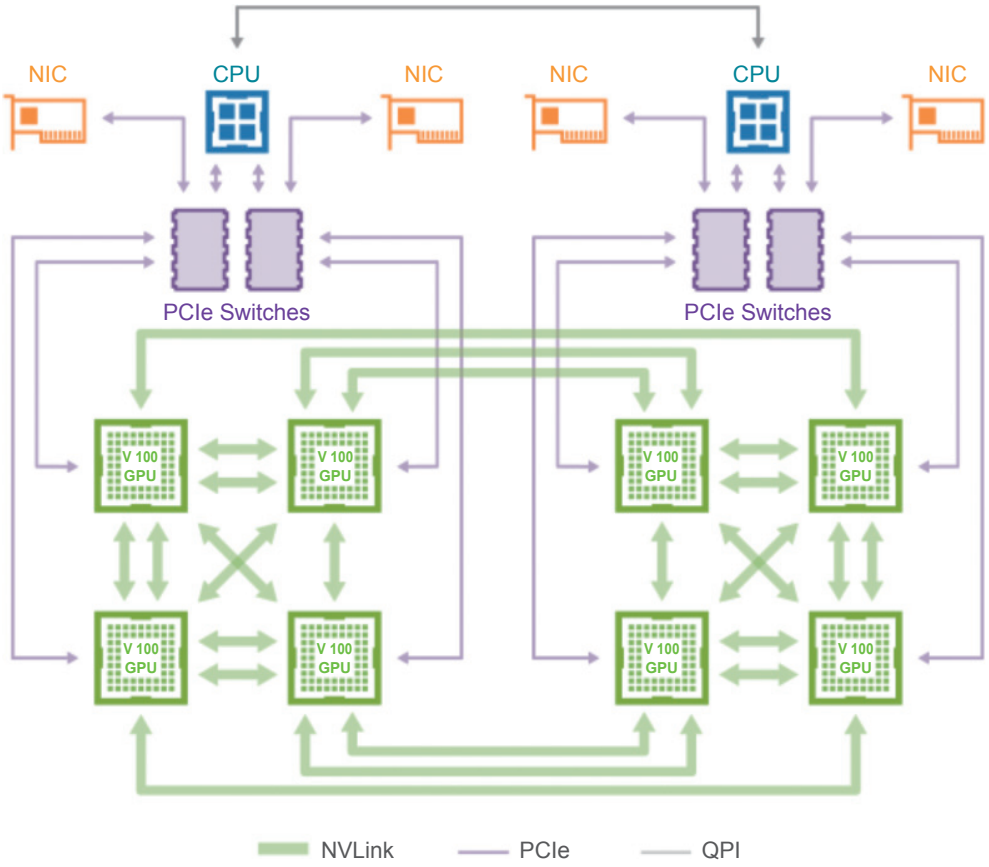


圖 4. DGX 及 HGX-1 系統架構。

我們研究團隊與業界合作，研究如何優化 HGX-1 的架構。在研究過程中，我們一度發現系統從 1 個 GPU 到 8 個 GPU 的硬體擴充，並不能讓效能呈線性成長，相較於 Google Tensorflow 官網所公布的 DGX-1 實測的效能指標，我們認為系統存在不可忽略的效能瓶頸。因此，我們使用 SOFA 裡各項效能資訊，對深度學習的計算過程，分解出如圖 5 的時序圖：

T_{cpu}：CPU 時間
T_{gpu_kernel}：GPU 核心時間
T_{gpu_comm}：CPU 通訊時間
T_{step}：單步訓練時間

以 VGG16 為例，我們透過 HGX-1 做網路訓練時採用複本式參數同步 (replicated-based parameters synchronization)，發現單步訓練時間為 10.24 秒，GPU 核心時間及通訊時間為 6.8 秒，這兩項扣除後，得到 CPU 時間佔比近乎 30%，在這般以 GPU/NVLink 為主角的計算任務中，這是出乎我們預料的。為了近一步了解 CPU 端的效能瓶頸所在，我們讓 SOFA 整合 Linux perf，動態地將分支預測誤判率 (branch-prediction miss rate) 、記憶體快取失效率 (cache-miss rate) 及匯流排周期數 (bus cycles) 等效能指標變化在時間軸上與函式群 (function swarm) 作套疊，進而發現快取失效率及匯流排周期數在模型權重更新 (weights update) 期間驟增。因此，我們檢查了系統計憶體的頻寬，得知由於記憶體安裝的方式非最優，而未能充分利用多通道記憶體技術提昇記憶體資料傳送效能，故在此案例中記憶體的頻寬只達到理想峰值的六分之一。舉一簡單例子來說明最優安裝的方式：當系統上配有雙通道且四條記憶體插槽 (依序為 1－4) 時，該主機板需要正確的安裝 1、3 或是 2、4 才能使用雙通道；若僅安裝 1、2 就會開啟單通道模式，純粹使系統記憶體容量增加，其頻寬不會

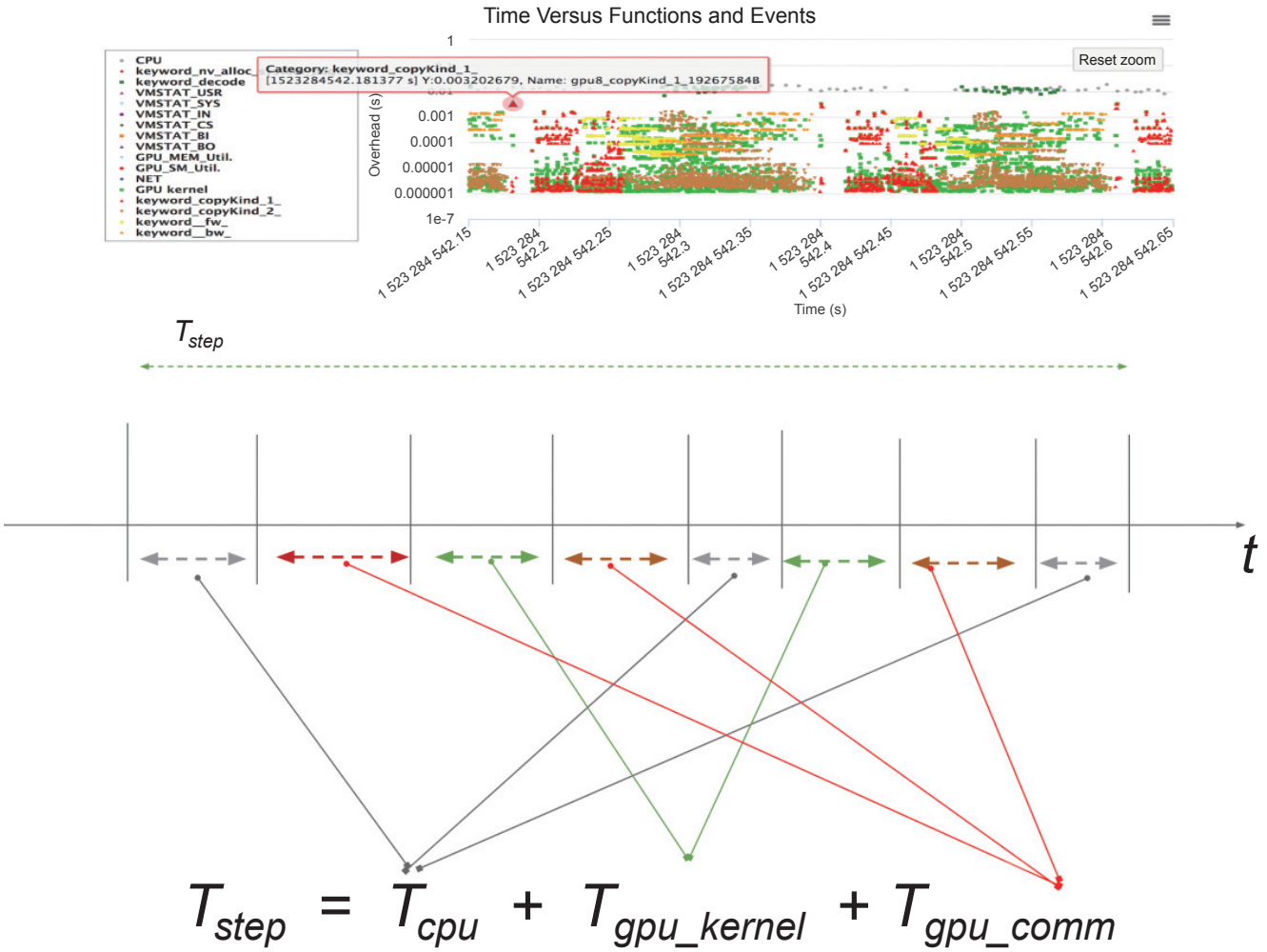


圖 5. 深度學習計算時序圖。

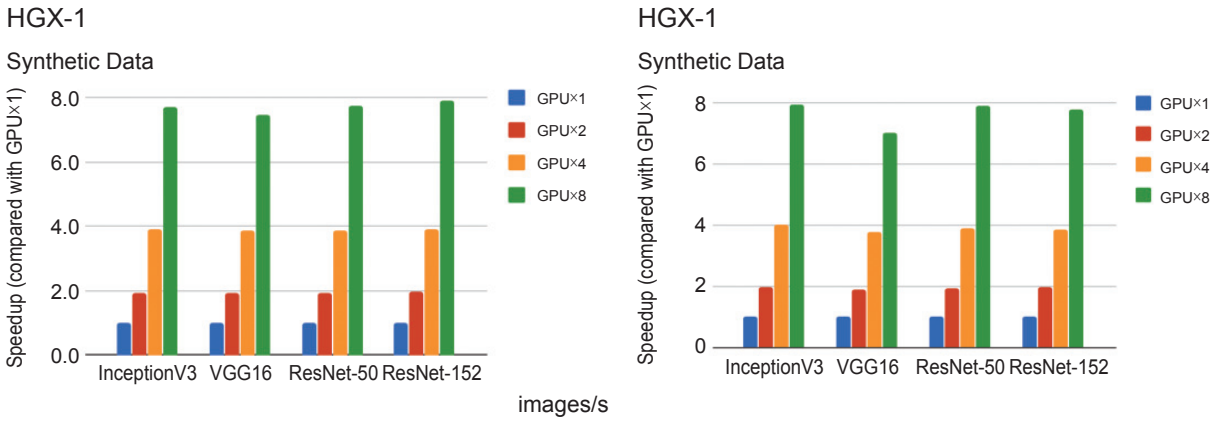
調升。經過重新記憶體安裝後，整體效能獲得顯著提昇。在圖 6 中，可看到 1－8 個 GPU 效能延展性 (performance scalability) 可媲美 DGX-1 執行 TensorFlow CNN Benchmarks 之表現。

・案例分析二：GPU 群集之網路通訊優化

Ring Allreduce 是 HPC 領域中相當經典的資料交換技術，它不但可降低資料頻寬需求，並利用雙向通道提高資料傳輸量，亦能夠有彈性的佈署在不同型態的網路拓撲上。知名深度學習平台 TensorFlow 將其納入後大幅提昇了 GPU 之間的數據傳輸速率，尤其是在 VGG16 及 ResNet152 這類具有眾多參數的神經網路模型裡，整體效能提昇更是顯著。然而，儘管 Ring Allreduce 算法理論上可以大幅舒緩通訊瓶頸，提升多 GPU 深度學習模型運算效率，但實際上該方法仍然很容易受制於環上單一邊上的最弱的通訊瓶頸 (weakest link)。以 HGX-1 多 GPU 之內部通訊為例，最佳化的 Ring Allreduce 的 ring，必由 NVlink 所生成，不應該經過 PCIe 匯流排，如圖 7 所示。

然而依照每個作業系統上 CUDA 驅動程式及 CUDA 執行期間函式庫的不同，使得 GPU ID 的列舉也會不同，若不特別進行設定，很容易讓某段資料傳輸走到頻寬較低的路徑。如圖 8 所示，GPU4 到 GPU5 及 GPU8 到 GPU1，本應有較佳的 NVLINK 走法，但因沒特別指定，而讓資料流經 PCIe bus 及 CPU Memory，使得效能下降。

在 SOFA 診斷出資料交換函式群 (communication swarm) 中有異常通訊模式後，我們將純粹以 NVLink 構成的網路拓撲表示成一個無向圖



HGX1-P100					DGX-1 P100				
Model	GPU×1	GPU×2	GPU×4	GPU×8	Model	GPU×1	GPU×2	GPU×4	GPU×8
InceptionV3	1.0	1.9	3.9	7.7	InceptionV3	1.0	2.0	4.0	8.0
VGG16	1.0	1.9	3.9	7.5	VGG16	1.0	1.9	3.8	7.0
ResNet-50	1.0	1.9	3.9	7.7	ResNet-50	1.0	1.9	3.9	7.9
ResNet-152	1.0	2.0	3.9	7.9	ResNet-152	1.0	2.0	3.9	7.8

圖 6. HGX-1 與 DGX-1 之效能延展性比較。

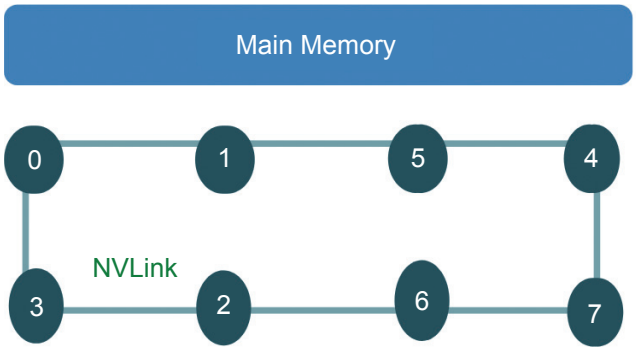


圖 7. 最優 All-Reduce Ring 之一。

(undirected graph)，在此無向圖中標示任一條通道傳輸資料所需的時間，並透過 CUDA_VISIBLE_DEVICES 指定 GPU 列舉順序，找到最優的環，可成功提昇多個網路模型之效能，如圖 9 之藍色長條明顯高於黃條，最顯著的是在執行 Vgg16 的時候高出 9.5%。

・案例分析三：深度學習之資料前處理優化

在深度學習計算中，CPU 端的效能議題常常會被忽略，資料前處理 (pre-processing) 就是個常常發生的例子。我們曾在 HGX-1 上採用多個 GPU 作深度學習計算，面臨到處理 ImageNet 資料集時，相較於處理合成資料，其效能的折損達到了 17%，相當於損失 1 至 2 片 GPU 效能的程度。簡言之，GPU 或是加速器的能力越強大，我們更必須關切在 CPU 上的前處理的效能。為了確切知道問題所在，我們透過上述的 SOFA 關鍵字函式分群功能，將「單一次迭代計算流程」切分為「資料前處理」(PRE)、「CPU 將資料傳送至 GPU 裝置 (H2D)」、「前向傳播 (FW)」、「後向傳播 (BW)」及「GPU 將資料傳送至 CPU 裝置 (D2H)」等階段，並輔以 perf 進行各個階段的熱點函式分析，遂而發現大量的 decode 及 DCT 等關鍵字佔了大部分的執行時間，如圖 10 所示。

此外，我們比較了在不同 CPU 核心數目時對於加速 PRE 階段進而提昇 throughput 的影響是顯著的，如圖 11 所見，推論出 PRE 階段可透過平行計算加速，只是因為某種因素導致效果不彰。

我們直觀地檢查向量指令集 (vector instruction) 的使用。進而發現 TensorFlow 在不同版本對於向量指令集有不同的支援度，採用 TensorFlow1.6 版 (官網稱該版本全面支援 AVX 指令集) 後，效能折損可從 17% 降至 6%。

Actual

Traffic Matrix (MB):

	HOST	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	GPU8
HOST	0	18473	0	0	0	19001	527	0	527
GPU1	1583	0	19001	527	527	0	0	527	0
GPU2	0	0	0	18473	0	0	0	0	0
GPU3	0	0	0	0	18473	0	0	0	0
GPU4	18473	0	0	0	0	0	0	0	0
GPU5	0	0	0	0	0	0	18473	0	0
GPU6	0	0	0	0	0	0	0	18473	0
GPU7	0	0	0	0	0	0	0	0	18473
GPU8	18473	0	0	0	0	0	0	0	0

Traffic Time Matrix (s):

	HOST	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	GPU8
HOST	0.00	1.74	0.00	0.00	0.00	1.79	0.05	0.00	0.05
GPU1	0.15	0.00	1.03	0.03	0.03	0.00	0.00	0.03	0.00
GPU2	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
GPU3	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
GPU4	1.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GPU5	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
GPU6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
GPU7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
GPU8	1.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Elapsed Step Comm. Time: 0.089329
Elapsed Step Kern. Time: 0.589300

圖 8. SOFA 通訊矩陣偵測出非最優 All-Reduce Ring 之列舉。

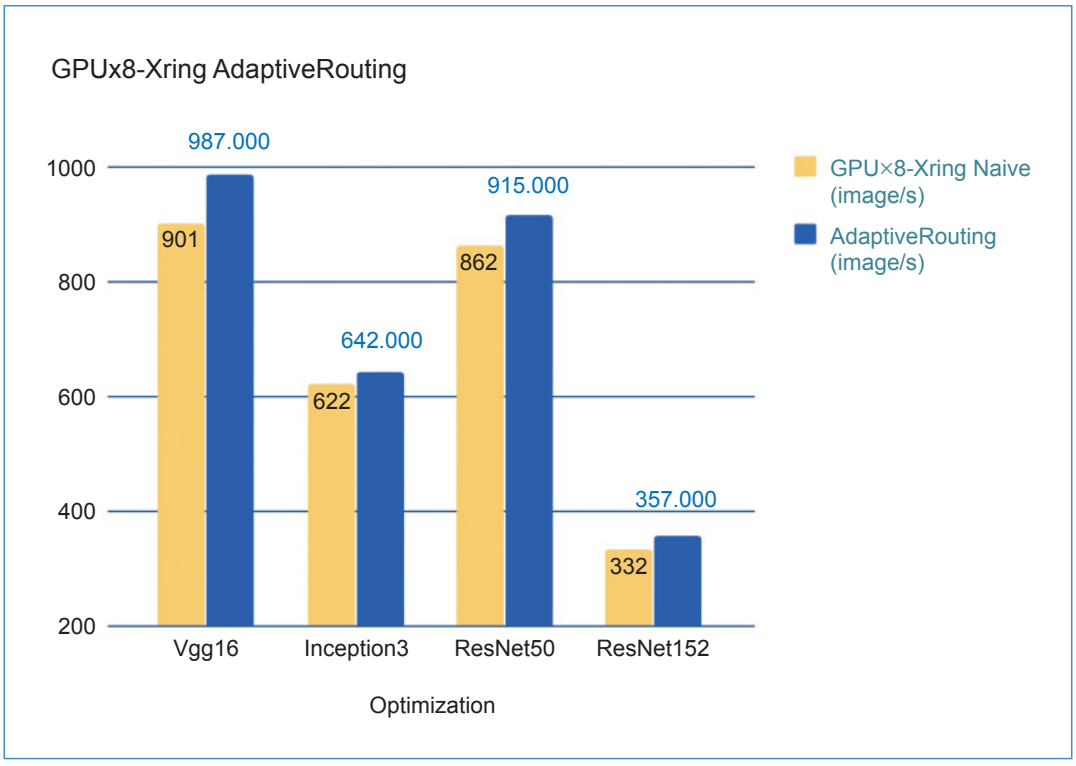


圖 9. 以優化 Ring Allreduce 作分散式訓練之效能提昇。

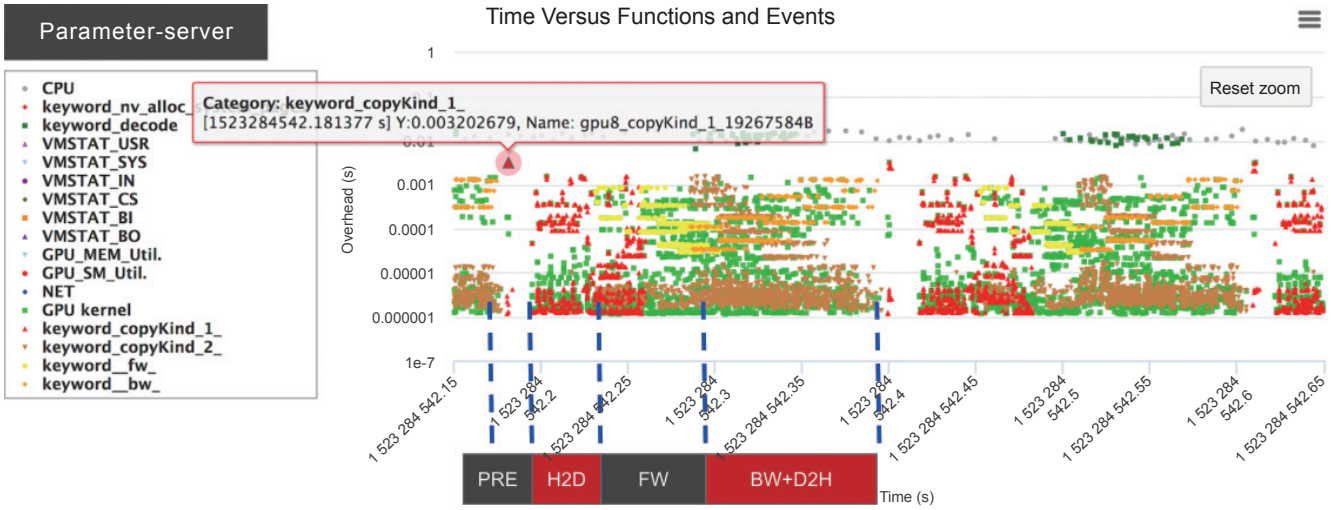


圖 10. 單一次迭代計算流程之函式分群。

六、結論

為了建構高效能分散式深度學習系統來加速現代人工智慧應用，本文提出之函式群分析法，已在實際深度學習任務上證明它的實用性，成功地在深層複雜軟體堆疊中，從不同面向定位出效能瓶頸，例如記憶體效能、群集之網路通訊模版以及資料前處理指令集優化。我們將此函式群分析法實作成為工具，命名為 SOFA，貢獻至開源軟體社群[19]。SOFA 可追蹤多樣化的效能指標，扮演著效能總監 (coordinator) 的角色，讓高效能異質計算平台上的應用得以在優化後發揮其潛力，加速各種深度學習演算法及高效能計算應用，提升學術或產業競爭力。

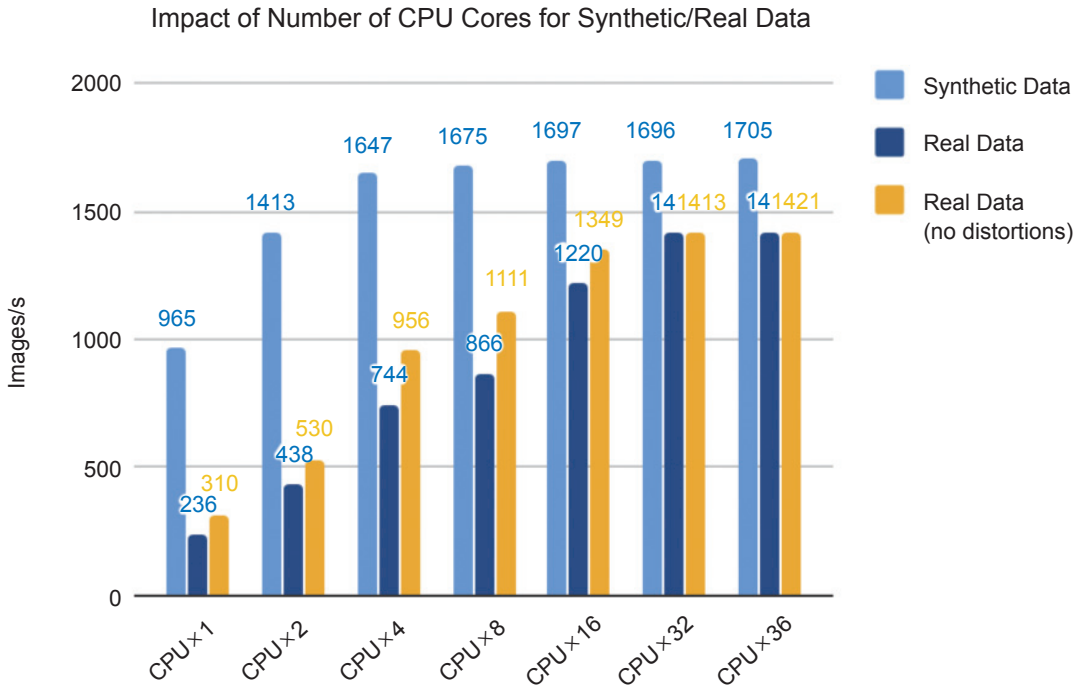


圖 11. 不同 CPU 核心數目時的 throughput 影響。

參考文獻

1. Kim, Heehoon, et al., “Performance analysis of CNN frameworks for GPUs”, *2017 IEEE International Symposium on Performance Analysis of Systems and Software*, 24-25 April, (2017).
2. Abadi, Martín, et al., “TensorFlow: A System for Large-Scale Machine Learning”, *OSDI'16 Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, 2-4 November, (2016).
3. Panda, Dhabaleswar K., and Xiaoyi Lu, “HPC Meets Cloud: Building Efficient Clouds for HPC, Big Data, and Deep Learning Middleware and Applications”, *10th IEEE/ACM International Conference on Utility and Cloud Computing*, 05-08 December, (2017).
4. Couturier, David, and Michel R. Dagenais, *Advances in Software Engineering*, **2015**, (2015).
5. Nadav Chachmon, et al., “Simulation and analysis engine for scale-out workloads”, *Proceedings of the 2016 International Conference on Supercomputing*, 01-03 June, (2016).
6. Seide, Frank, and Amit Agarwal, “Cntk: Microsoft's open-source deep-learning toolkit”, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17 August, (2016).
7. Chen, Tianqi, et al., “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems”, arXiv:1512.01274, (2015).
8. Jouppi, Norman P., et al., “In-datacenter performance analysis of a tensor processing unit”, *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 24-28 June , (2017).
9. Caulfield, Adrian M., et al., “A cloud-scale acceleration architecture”, *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 15-19 October, (2016).
10. Brendan Gregg, *Communications of the ACM*, **59** (6), 48 (2016).
11. James Reinders, *VTune performance analyzer essentials*, Intel Press (2005).
12. Nethercote, Nicholas, and Julian Seward, *ACM Sigplan notice*, **42** (6), (2007).
13. Weaver, Vincent M, “Linux perf_event features and overhead.” The 2nd International Workshop on Performance Analysis of Workload Optimized Systems, *FastPath*, **13**, (2013).
14. Couturier, David, and Michel R. Dagenais, *Advances in Software Engineering*, **2015** (2015).
15. Brunst, Holger, et al., “Performance optimization for large scale computing: The scalable VAMPIR approach”, *International conference on computational science*, 28-30 May, 2001.
16. Mohammad, Alian, et al., “dist-gem5: Distributed simulation of computer clusters”, *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 24-25 April, 2017.
17. Tu, Chia-Heng, et al., *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, **19** (2), 10 (2014).
18. Nadav Chachmon, et al., “Simulation and analysis engine for scale-out workloads”, *Proceedings of the 2016 International Conference on Supercomputing*, 01-03 June, (2016).
19. Please refer to the web site: https://github.com/cyliustack/sofa

---

劉政岳先生現為國立臺灣大學資訊工程系博士候選人。

Cheng-Yueh Liu is currently a Ph.D. candidate in the Department of Computer Science and Information Engineering at National Taiwan University.

洪士灝先生為美國密西根大學電腦科學及工程學博士，現為國立臺灣大學資訊工程學系教授。

Shih-Hao Hung received his Ph.D. in computer science and engineering from Michigan University, USA. He is currently a professor in the Department of Computer Science and Information Engineering at National Taiwan University.